Renaming Collection Design

Goal

Provide a way change a collection identifier that may be referred to by granules within the collection.

Traceability

⚠ CMR-2334 - JIRA project doesn't exist or you don't have permission to view it.

Background

Several providers have attempted to change or requested the ability to change a collection's short name, version id, or entry title. Some providers would like to do this as part of their data reconciliation. This is currently difficult to do since granules can refer to a collection using any of those ids. The current behavior is to reject an update to any of those fields if the collection has any granules. Allowing the provider to directly update those ids would mean we would have to modify their granules' metadata. One of the original requirements of the CMR (⚠ CMR-108 - JIRA project doesn't exist or you don't have permission to view it.) was that the CMR shouldn't use the dataset id as the primary key so that it could be changed. We don't use the dataset id as the primary key but we still have the original problem (changing it requires deleteing and reingesting all granules) because the granule metadata still refers to it.

Requirements

1. Must be asynchronous
    a. There are too many granules to update in a synchronous single request and in one transaction. It would have to be done asynchronously and not within a single transaction. This means that for a time the data in the CMR will contain granules which refer to the old collections id.
2. Must be resilient to failures.
    a. Once a provider kicks off a rename there must be a very high probability that the rename will complete without issue. We don't want Operations to have to intervene often or providers have to ping operations if there's an issue
3. Must finish in a reasonable amount of time.
4. Providers would need a way to check on the status of the job.
    a. Since this is an asynchronous action a provider would like to know when it is complete.
5. Must handle concurrent requests to rename the same collection.

High Level Options

There are multiple solutions that could work to solve this issue.

1. Ingest detects when a provider sends a collection that updates one of the identifier fields and automatically updates all granules in the collection.
2. Ingest offers an explicit endpoint where a provider can submit a request to update the collection identifiers.
3. Providers submit a request to CMR Administration to rename a collection. The admin uses the CMR bootstrap application to perform the rename and update all the granules.

Options 1 and 2 are very similar and would have similar implementation. They have the benefit that a provider can make the change on their own. They don't need to coordinate with anyone to make the update. The downside with these is the implementation difficulty which I'll document below.

Option 3 is the easiest to implement. It's going to be very similar to other activities the bootstrap application can perform. The downside is that it requires a provider to coordinate with a CMR administrator to perform the update.

When is the Collection Modified?

Note that with any of the options the first step is updating the collection to change it's identifier. That would be saved as a new revision in Metadata DB.

What if granules updates are sent during the asynchronous updates?

Granule updates sent from the provider that reference the original collection identifier would be rejected. The first step is updating the collection so any granule updates sent after that would be validated against that collection. Granule updates sent with the new collection identifier would be accepted. There is a potential race condition in that a provider sent update could be overwritten by the asynchronous process that's updating granule metadata. We will avoid this race condition by sending in the expected revision id when updating the granule metadata. If new granule metadata is sent after the async process retrieves the metadata but before the update the revision will have been incremented. Metadata DB will reject the update in the async process because it's expected revision id has already been used. This would trigger a failure in the async process. The async process would either need to explicitly handle this and retry or have a general retry capability.

Provider Initiated Design (Options 1 and 2)

This option describes an implementation that would allow a provider to trigger a rename and handle it completely themselves. This requires a more robust implementation than renaming kicked off by an administrator.

Meeting the Requirements

1. Must be asynchronous
    a. We would need some way internally to submit job processing requests and track their status.
2. Must be resilient to failures:
    a. In order to provide this capability processing of a rename would need to be initiated via the processing of a message from the message queue. Message processing can handle failures and trigger retries.
    b. Retries also mean that the process needs to be idempotent. A previous attempt to rename granules may have finished partially.
3. Must finish in a reasonable amount of time
    a. In order to support changing collection ids in large collections we may have to distribute the work across multiple hosts. This means the initial message to rename granules would likely just find granules to rename and submit batches of them as secondary messages to process. Multiple hosts can participate by listening for those messages and processing them. Separate messages could be processed in parallel.
4. Providers would need a way to check on the status of the job.
    a. We must have job status tracking to support this. We would need to store that data somewhere, update it during processing, and provide an API to retrieve status information.
5. Must handle concurrent requests to rename the same collection.
    a. Either handle it through versioning of the id or detect that a job is still running for that collection and prevent the update.

Renaming via Administrator Action (Option 3)

This option describes an implementation that utilizes CMR Administrators to perform the rename using the Bootstrap application. This is a less robust solution (easier to implement)

Meeting the Requirements

1. Must be asynchronous
    a. Providers would email CMR Ops to ask for a rename. CMR Ops would perform the rename using the bootstrap API which could rename the collection and all of the granules asynchronously.
2. Must be resilient to failures:
    a. There's less of a failure resiliency requirement with this at the code level. CMR Ops can monitor the rename via CMR Bootstrap logs. Any problems encountered would be investigated and the process could be restarted.
3. Must finish in a reasonable amount of time
    a. Bootstrap can run this on a single host. Bootstrap embeds the metadata db and indexer applications which removes the need for slow network IO. The Bootstrap application uses core.async to execute parts in parallel which should give reasonable performance.
4. Providers would need a way to check on the status of the job.
    a. Providers can email CMR Ops to check on the status of a rename. CMR Ops will email the provider when the rename is complete.
5. Must handle concurrent requests to rename the same collection.
    a. The Bootstrap application would not be able to handle concurrent requests but CMR Ops would know not to do this. If a provider makes concurrent requests (emails) to CMR Ops they would know to wait for the previous request to complete before continuing.

Bootstrap will asynchronously create new revisions of granules that have updated identifiers

1. Find the latest revision of all granules in a collection
2. Filter out tombstones
3. Update the metadata to use the new identifier
4. Save updated revision to metadata db and index it

Core Async Process

Bootstrap uses the Clojure core.async library that allows defining concurrent processing steps that communicate via channels. The following steps define each concurrent processing step and what happens.

Find batches of granule concept ids in parent collection

We'll use a work table to identify the set of granules to update. The rename_parent_coll_work table will contain granule concept ids and the parent collection id being worked. The table can be used in parallel for different collections since the collection id provides uniqueness. The table should be cleared with the collection id when starting a run to remove rows from previous runs. The table should also be cleared after the process has completed successfully.

Populate the work table with the following sql:

```
insert into rename_parent_coll_work values (select distinct concept_id, parent_collection_id from granules
where parent_collection_id = ?)
```

Select batches of granule concept ids from the table to use. Put each batch on the channel.

Find latest revisions of a batch of granule concept ids

Use existing functions that take a set of concept ids and return latest concepts. We'll process each batch of concept ids and place a batch of the retrieved latest revisions of the concepts on the next channel for processing.

Process Batch

- Filter items from each batch
  - Filter out tombstones
- Update the metadata
  - This will need to apply an update to the granule metadata to replace the existing id with a new one.
  - I would suggest we use clojure.data.xml to parse, modify the data structure, and then write it back out. We'll need to do that with both ECHO10 and ISO SMAP granules.
- Put each individual granule on a new channel

Save and index each updated granule

We'll save and index each granule from the channel using the cmr.bootstrap.data.util/save-and-index-concept function.

Recommended Option

We recommend using the Renaming via Administrator Action option as the initial implementation. It can be implemented in under a week and the code will be reusable in a more robust implementation later. The Provider Initiated Design would likely be an epic of issues to implement and may take 2 - 3 weeks to implement.

Error rendering macro 'pageapproval' : null